NAS Experiences of Porting CM Fortran Codes to HPF on IBM SP2 and SGI Power Challenge

Report NAS-95-010, April 1995

Subhash Saini[1]
Numerical Aerodynamic Simulation Facility
NASA Ames Research Center
Mail Stop T 27A-1
Moffett Field, CA 94035-1000

Phone:415-604-4343
Fax:415-604-3957
E-mail:saini@nas.nasa.gov

Abstract

Current Connection Machine (CM) Fortran codes developed for the CM-2 and the CM-5 represent an important class of parallel applications. Several users have employed CM Fortran codes in production mode on the CM-2 and the CM-5 for the last five to six years, constituting a heavy investment in terms of cost and time. With Thinking Machines Corporation's decision to withdraw from the hardware business and with the decommissioning of many CM-2 and CM-5 machines, the best way to protect the substantial investment in CM Fortran codes is to port the codes to High Performance Fortran (HPF) on highly parallel systems. HPF is very similar to CM Fortran and thus represents a natural transition.

Conversion issues involved in porting CM Fortran codes on the CM-5 to HPF are presented. In particular, the differences between data distribution directives and the CM Fortran Utility Routines Library, as well as the equivalent functionality in the HPF Library are discussed. Several CM Fortran codes (Cannon algorithm for matrix-matrix multiplication, Linear solver *Ax=b*, 1-D convolution for 2-D datasets, Laplace's Equation solver, and Direct Simulation Monte Carlo (DSMC) codes have been ported to Subset HPF on the IBM SP2 and the SGI Power Challenge. Speedup ratios versus number of processors for the Linear solver and DSMC code are presented.

1. Subhash Saini is an employee of Computer Sciences Corporation. This work was funded through NASA contract NAS 2-12961.

1 Introduction

The Numerical Aerodynamic Simulation (NAS) Program, located at NASA Ames Research Center, is a pathfinder in high performance computing for NASA and is dedicated to advancing the science of computational aerodynamics. One key goal of the NAS organization is to demonstrate by the year 2000 an operational computing system capable of simulating an entire aerospace vehicle system in one to several hours. It is currently projected that the solution of this Grand Challenge problem will require a computer system that can perform scientific computations at a sustained rate approximately 1000 times faster than 1990 generation supercomputers. Most likely such a computer system will employ hundreds or even thousands of powerful RISC processors operating in parallel.

NAS currently supports two CRAY C90 systems: a TMC CM-5; the HPCC cluster comprising 16 SGI Power Challenges; 2 HP PA 7200; 2 IBM RS 6000/590 workstations; an IBM SP2 and Intel Paragon; 250 SGI workstations (R2000, R3000, and R4000), and 95 Sun workstations. The CM-5 was decommissioned on March 31, 1995 and the Intel Paragon is scheduled to be decommissioned on July 31, 1995.

High performance Fortran (HPF) [1-4] addresses the need of a large number of parallel computers users. NAS has a large base of CM-5 users and a significant number of production codes with good performance being run on it. We know that other large CM-5 and parallel computer centers such as NCAR, AHCRC, LANL are faced with same problem. We believe that this is also true for many large commercial and government sites. Most of the codes on CM-5 are written in CM Fortran, and users are extremely reluctant to switch to message-passing paradigms.

This paper is organized as follows: Section 2 gives taxonomy of various programming models available on parallel computers. Section 3 gives the reasons for using HPF at NAS. Section 4 presents the general characteristics of CM Fortran codes and their similarities with Fortran 90 and HPF. Section 5 discusses the various conversion issues involved in porting CM Fortran codes to HPF. Section 6 deals with programming tools available in the programming environment of CM Fortran and HPF. Section 7 gives brief a overview of currently available HPF compilers. Section 8 presents results of porting CM Fortran codes to HPF. Deficiencies of current HPF compilers are given in Section 9. Conclusions are presented in Section 10.

2 Taxonomy of Programming Models

Currently, the following programming models are in common use on various highly parallel systems:

2.1 Message Passing Paradigm

Message passing is the most common programming model available on highly parallel systems. One of the oldest message-passing libraries is NX on Intel iPSC/860 [5] and Intel Paragon [6]. The syntax requires the programmer to explicitly decompose arrays and communicate data. Parallel Virtual Machine (PVM) [7] and Message Passing Interface (MPI) [8] are considered two *de facto* standards. MPI is expected to become a standard. However, native MPI is yet not supported on the IBM SP2 [27] or the CRAY T3D.

Explicit message passing is not viewed by most of the users as a long-term solution to the scientific community's programming needs but merely something necessary in the short term until parallel languages and efficient compilers are developed and commercially available. This will obviate the need for users to explicitly specify and manage the interprocessor communication in their applications. It is widely believed that HPF offers the best hope for achieving this objective.

2.2 Data Parallel Paradigm

In data parallel model, the same operation is performed on many data elements by many processors simultaneously. Global data structures are distributed uniformly and processing power follows data. Conceptually, each processor computes only one element of each dataset. Data parallelism is a simpler programming paradigm that scales naturally as data grows. Some commonly used data parallel languages are as follows:

2.2.1 CM Fortran

CM Fortran [9] is a proprietary language of Thinking Machines Corporation. It is implemented as a data parallel language with Fortran 90 [10] arrays as its parallel data structures. All Fortran 77 operations and intrinsic functions are extended to act point-wise on arrays. New intrinsic functions are provided for constructing and transforming arrays. In addition, CM Fortran also has a `FORALL` statement and a `FORALL` construct. The application programmer sees only a single thread of control and views data globally. Distribution of data and computation, communication, and synchronization are managed automatically. The programmer can modify data distribution or invoke special communication patterns. The programmer has to be very careful about the pattern of interprocessor communication and to limit it to the near-neighbor, if possible. The interprocessor communication is typically coded using intrinsic functions such as `CSHIFT` and `EOSHIFT`. In many CM Fortran codes, data parallel syntax can be difficult to decipher, especially when the code makes extensive use of `CSHIFT` and `EOSHIFT`. Array elements not explicitly distributed are mapped according to an implementation-dependent default distribution called the canonical distribution (see Section 5.4 for details)

2.2.2 High Performance Fortran

HPF, the premier high-level language for parallel programming is based on a number of existing languages such as CM Fortran [9], Fortran 90 [10], Fortran D [11], and Vienna Fortran [12]. HPF allows users to write a single parallel program that can execute on both distributed-memory (IBM SP2) and shared-memory machines (DEC TurboLaser or SGI Power Challenge XL). HPF eliminates the complex, error-prone task of explicitly telling the compiler how, where, and when to pass messages between processors on distributed-memory machines, or when to synchronize processors on shared-memory machines. HPF codes can execute a single program on a distributed-memory cluster, a single Symmetric Multiprocessing Processor (SMP), or a cluster of SMPs.

Extensions incorporated into HPF provide a means for explicit expressions of parallelism and data mapping. These extensions are (a) directives advising the compiler how data objects should be allocated to processor memories, and (b) new language features such as the `FORALL` statement and `FORALL` construct borrowed from CM Fortran. HPF uses compiler directives `PROCESSORS, ALIGN, DISTRIBUTE,` and `TEMPLATE` to map array elements to abstract processors in a two-level mapping: (i) The array elements are first aligned with an abstract index space called a template [22]. (ii) The template is then distributed onto a rectilinear arrangement of abstract processors.

2.2.3 Cray Research Adaptive Fortran (CRAFT)

CRAFT [13-15] derives its features from Fortran 77, Fortran 90, Fortran D, and HPF. CRAFT has the following three directives: (a) Data Sharing Directives: Data sharing directives distribute data, such as an array, over the memory of the processor elements (PEs). The CRAY T3D system distinguishes between data objects that are private to a task and those that are shared among all tasks; (b) Work Sharing Functions and Directives: CRAFT provides an implicit programming method called work sharing. Work sharing directives divide the work by distributing iterations across all available processors. Also, functions are available for determining if a code is in a parallel or sequential region; (c) Synchronization Functions and Directives: CRAFT provides synchronization functions and directives to coordinate the processor tasks: Barriers, ATOMIC UPDATES, LOCKS, CRITICAL sections, EVENTS, and EUREKA functions. HPF emphasizes on its data distribution mechanisms whereas CRAFT emphasizes on work distribution mechanism. The drawback of CRAFT is that codes written using CRAFT-specific directives are not portable to another machine.

2.2.4 Silicon Graphics Power Challenge XL

SGI's Power Fortran Accelerator (PFA) is a source-to-source optimizing Fortran preprocessor that discovers parallelism in Fortran code and converts those programs to parallel code [15-17]. Cray CDIR$ directives are accepted. One can also insert compiler directives for parallelization. Interprocessor communication is handled by cache coherent hardware. Parallelization can be achieved (i) Fully automatically: done by PFA, (ii) Semi-automatically: User manually inserts directives, (iii) Manually: User inserts library/system level call. CPUs communicate and synchronize through coherent shared memory. The drawback is that codes written using SGI-specific directives are not portable to an other machine.

2.2.5 Convex Exemplar SP1000

Semiautomatic parallelization in conjunction with the hardware's ability to dynamically distribute array elements provides a shared memory paradigm for existing Fortran 77 codes [18]. Convex-specific directives are available to aid the compiler in achieving parallelization. Application developers can use these directives to force the parallelization of data-dependent loops and can code explicit synchronizations among the threads to correctly handle the dependencies. More advanced users can use features supporting explicit parallel thread management, explicit synchronization, and explicit data distribution. The drawback is that codes written using Convex-specific directives are not portable to another machine.

2.2.6 Fujitsu VPP500

Parallel processing is performed using VPP Fortran compiler directives to Fortran 77 [19]. It is based on index decomposition and distribution onto processors while maintaining the correspondence between partitioned DO loops and distributed data. Here also, the drawback is that codes written using these Fujitsu-specific directives are not portable to another machine.

2.2.7 DEC Alpha Server 8400 5/300 (TurboLaser)

The control parallelism can be handled directly by the KAP parallel-processing directives. A user can also insert KAP-specific directives for parallelization. The drawback is that codes written with KAP-specific directives are not portable to another machines.

2.3 Explicit Shared Memory Model (CRAY T3D and Meiko)

In the explicit shared-memory model, the user is responsible for explicitly managing the data transfers and communication by explicitly specifying two addresses: a local address, and a remote address, which is really a tuple (PE number, local address). This is done by shared-memory access libraries interface. This interface allows users to address the global memory explicitly, without the compiler needing to know how to compute glob addresses. The two functions (shemm_get and shemm_put) are used to read from/to any remote PE and map directly onto the low level CRAY T3D concepts of loads and stores. These functions have been used to optimize all the existing CRAY T3D benchmarks including NAS Parallel Benchmarks [20]. Meiko also has implemented this programing model, which is difficult for users to implement and is not portable to another computer.

3 Why HPF at NAS?

The NAS CM-5 was decommissioned on March 31, 1995. Several users had been using CM Fortran on CM-2 and CM-5 for the last 5 years; thus, there is a heavy investment in CMF codes [21]. Since CM Fortran very close to High Performance Fortran (HPF), there is a natural transition from CM Fortran to HPF. Users prefer HPF to Message Passing Paradigm because of programming ease. The availability of the excellent CM Scientific and Subroutines Library (CMSSL) [25] on the CM-5, and almost negligible parallel mathematical and scientific libraries using the message-passing paradigm, speaks of the ease of developing applications and scientific libraries in this data parallel language. There is a hope of rich mathematical and scientific libraries in HPF. In addition, Subset HPF compilers are available on the NAS IBM SP2 and the NAS High Performance Computing and Communication Program (HPCCP) cluster; thus, CM-5 users can port their codes to HPF.

4 Characteristics of CM Fortran Codes

A quick look at any CM Fortran code reveals that it is comprised mostly of the following statements, structures, and directives:

a. Array syntax, which is a part of Fortran 90 language.
b. `WHERE` statement and `WHERE` construct, which are parts of Fortran 90 language.
c. Intrinsic shift functions: `CSHIFT` and `EOSHIFT` which are parts of Fortran 90 language.
d. `FORALL` statement and `FORALL` construct, which are parts of HPF specification.
e. `CMF$ LAYOUT` directives for data distribution - like HPF compiler directives.
f. `CMF$ ALIGN` directive for data alignment - like HPF directive `ALIGN`.
g. CM Fortran utility routines with equivalent routines in HPF Library.
h. Parallel I/O that have no counterpart in HPF, as there is no parallel I/O in HPF.

It is clear that, except for parallel I/O, porting CM Fortran codes to HPF is a natural transition [21].

5 Porting CM Fortran Codes to HPF

In this section, we discuss the various conversion issues involved in porting CM Fortran codes to HPF [21].

5.1 Data Distribution Directives

CM-5's Run Time System (RTS) distributes array elements among the processors available to the program. The particular distribution of an array is called its *layout* [23]. The layout chosen by the RTS without intervention from the user is the canonical layout or canonical distribution. The main reason to use layout directives is to improve performance over what a user cannot obtain using the canonical layout. The CM Fortran data distribution directives are much simpler than ones available in HPF. For the CM Fortran on the CM-5, there are two layout directives: `LAYOUT` and `ALIGN`. For CM Fortran running on the CM-2, in addition to these two there is a third directive called `COMMON`.

5.1.1 LAYOUT Directive

In CM Fortran, an arbitrary numbers of arrays of an arbitrary number of dimensions can be specified in the same `CMF$ LAYOUT` directive as shown in Figure 1. Note "weight" 2 in `2:NEWS` in

Example: 1 (a)

```
   INTEGER, PARAMETER :: N=256
   REAL*8, ARRAY(n,n,n) :: a
CMF$ LAYOUT a(:SERIAL,2:NEWS,:NEWS)
```

Example: 2 (a)

```
   INTEGER, PARAMETER :: m=128
   REAL*8, ARRAY(m,m):: x
   REAL*8, ARRAY(m):: y
CMF$ LAYOUT(:SERIAL,2:NEWS),y(:NEWS)
```

Figure 1: Use of layout directive in CM Fortran.

Example: 1 (b)

```
   INTEGER, PARAMETER :: N=256
   REAL*8, DIMENSION(n,n,n):: a
CHPF$ DISTRIBUTE a(*,BLOCK,BLOCK)
```

Example: 2 (b)

```
   INTEGER, PARAMETER :: m=128
   REAL*8, DIMENSION(m,m):: x
   REAL*8, DIMENSION(m) :: y
CHPF$ DISTRIBUTE x(*, BLOCK)
CHPF$ DISTRIBUTE y(BLOCK)
```

Figure 2: Use of distribute directive in HPF.

layout directive. There is no concept of weight in HPF. However, HPF does not allow specifying more than one array in the same line of `CHPF$ DISTRIBUTE` due to the stringent attributed form of the distribution directive as shown in Figure 2. In summary, `SERIAL` translates into `*` and `NEWS` translates into `BLOCK`.

Many CM Fortran codes developed on the CM-2 use the following COMMON directives:

```
CMF$ COMMON FEONLY /blk1/a(100,100)
CMF$ COMMON CMONLY /blk2/ b(100,100)
CMF$ COMMON INITIALIZED/blk3/c(100,100)
```

On the CM-2, these directives are meant to specify the home of a given array. The front-end is the default home for array a(100,100) in `COMMON` block `blk1`. The CM is the default home for array b(100,100) in `blk2`. Array c(100,100) in `blk3` is stored on CM and has front-end memory allocated for initialization in a `BLOCK DATA` unit. These `COMMON` directives should remain untouched and the compiler will ignore them.

5.1.2 ALIGN Directive

As discussed in the previous section, a LAYOUT directive can align a vector only with the first row or column of array. However, an ALIGN directive can match the vector with any row or column of array. In CM Fortran, for statically allocated arrays: (a) conformable arrays with canonical layouts are aligned and (b) conformable arrays with identical noncanonical layouts are aligned. The ALIGN directive can enhance the performance of an application by eliminating unnecessary data communication when operations are performed on multiple arrays that would not normally be aligned. The use of ALIGN directive in CM Fortran and in HPF is shown in Figures 3 and 4, respectively. Many CM Fortran codes rely on the canonical alignment and do not use any ALIGN directives. However, such codes, when ported to HPF must use the ALIGN directive for better performance as there is no canonical alignment in the current implementation of HPF compilers. It is recommended that HPF compilers should be free to use a canonical `align` when none is specified by the user.

5.1.3 TEMPLATE Directive

In HPF, a template is an abstract space of indexed positions [22] - an “array of nothings” as

To align vector b with first row of matrix a

```
   INTEGER, PARAMETER :: N=256
   REAL*8, ARRAY(n,n) :: a
   REAL*8, ARRAY(n) :: b
CMF$ ALIGN b(j) WITH a(1,j)
```

Figure 3: Use of ALIGN directive in CM Fortran.

```
   NTEGER, PARAMETER :: N=256
   REAL*8, DIMENSION(n,n) :: a
   REAL*8, DIMENSION(n) :: b
CHPF$ ALIGN b(j) WITH a(1,j)
```

Figure 4: Use of ALIGN directive in HPF.

opposed to, say, an "array of reals." One can use a template as an abstract *align-target* that can then be distributed. There is no concept of TEMPLATE in CM Fortran. In some HPF compilers (*e.g.*, in pghpf), if arrays are not aligned to the same template, the compiler does not recognize efficient communication patterns. However, when arrays are aligned to a common template the compiler generates efficient code. An example illustrating the use of template is shown in
Figure 8.

5.2 CM Fortran Utility Routines

CM Fortran utility routines provide convenient access from CM Fortran to the capabilities of lower-level CM software [24]. Purpose of this was to provide a functionality and performance beyond what was available from the compiler. With maturing of the compiler most of these functionalities were included in the compiler. Nevertheless, many CM Fortran codes even today rely significantly on these utility routines. Most of these routines have equivalent functionality in HPF Library routines of HPF.

5.2.1 Parallel Prefix Operations

Many CM Fortran codes use subroutines from CM Fortran Utility Library to perform prefix operations, or scans on one axis of an array using

```
CALL CMF_SCAN_combiner(DEST, SOURCE,SEGMENT,AXIS,DIRECTION,INCLUSION,SEGMENT_MODE, MASK)
```

where combiner = {copy,add,max,min,ior,iand,ieor}. There are equivalent routines in HPF library procedures. The only difference between these two set of routines is in segment representation. For example, in CM Fortran Utility Library, segment vector (/T,F,T,F,F,F,T,F,F/)represents three segments of size 2, 4 and 3. However, in HPF the equivalent segment is given by (/T,T,F,F,F,F,T,T,T/. Conversion between these two forms is easy using the parity-prefix function.

5.2.2 Parallel I/O

On CM-5, the usual Fortran OPEN, WRITE, and READ statements do very fast parallel I/O for large parallel arrays using Scalable Disk Array (SDA). There is no provision for parallel I/O in HPF. Parallelism in I/O is necessary for performance of I/O dominated codes. The I/O dominating codes such as out-of-core solvers written in CM Fortran will not perform well in HPF unless parallel I/O is included in the HPF.

5.2.3 Random Numbers

On CM-5, CMSSL provides two versions of a lagged Fibonacci pseudo-number generator [25]. The implementation requires that every processor maintain a state table that is massaged on each call to produce the next element in the output sequence. The two versions differ in the manner of allocating these tables. For generating random numbers in CM Fortran, a subroutine called CMF_RANDOM is used. The equivalent routine for generating random numbers in Fortran 90 is

RANDOM_NUMBER. There is no equivalent to the fast version of random generators in Fortran 90 or HPF. It is recommended that the next release of HPF include fast random generator routine. The current implementation of intrinsic subroutine RANDOM_NUMBER is serial with very low performance.

5.2.4 Sorting

Many CM Fortran codes use CMF_ORDER utility routine [24]. This routine places the rank of each element along the specified axis of a source array into the corresponding element of the destination array, under the control of a logical mask and is used as

```
        CALL CMF_ORDER(DEST,SOURCE,AXIS,MASK)
```

This routine returns a rank vector. The routine close to this functionality is GRADE_UP routine from the HPF Library. However, it requires a permutation of the input array indices that describes the ordering.

5.3 FORALL Construct

Many CM Fortran codes use the FORALL construct [26]. Currently, Subset HPF compilers don't provide this construct. Therefore the FORALL construct in CM Fortran codes needs to be converted to FORALL statements. Many CM Fortran codes could not be ported to HPF because the FORALL construct is not included with current HPF compilers.

5.4 Canonical Array Layout

In CM Fortran, by default, for every CM fortran array the CM Run-Time System (CMRTS) uses a canonical data distribution. It is not necessary to use layout directives for CM Fortran arrays. The CMRTS uses block layout of elements as opposed to cyclical layout. In HPF, compilers don't distribute arrays by default and, in fact, they are replicated. For porting CMF codes to HPF, it is necessary to insert HPF mapping directives to distribute the data. Code of Figure 7 relies on canonical distribution feature of CM Fortran and therefore does not use layout directive. However, port of the code in HPF is shown in Figure 8 and uses TEMPLATE, ALIGN, and DISTRIBUTE directives of HPF. It is recommended to vendors to include canonical alignment and canonical distribution as default in HPF compilers.

5.5 CMSSL Library

Most of production codes running on CM-2 or CM-2 use routines from the CMSSL [25]. Commonly used CMSSL routines on CM-5 at NAS are FFTs, block and scalar pentadiagonal solvers, and block tridiagonal solvers. None of these routines is available in HPF on any machine. One software vendor has existing message-passing library modelled on IMSL. They are just starting on implementation of a prototype F77_LOCAL extrinsic as a model around which a more extensive tuned library can be developed. It's not clear at present whether this effort will be successful. At NAS, an effort is underway to develop these routines in HPF [39]. It looks as though the big obstacle in porting CM Fortran codes of CM-5 to HPF is the lack of CMSSL equivalent routines in HPF. The 3-D FFT (FT) CM Fortran code from the NAS Parallel Benchmarks could not be ported to HPF because of the absence of parallel 3-D FFT that can be used from HPF. This obstacle must be overcome soon to facilitate the easy porting of CM Fortran codes on CM-5 to HPF.

5.6 Communication Compiler

Many CM Fortran codes use a communication compiler to precompute the communications patterns [25]. This functionality is provided in CMSSL. There are no equivalent routines to replace communication compiler routines. Thus, many CM Fortran codes using a communication compiler when ported to HPF will run very slowly. This functionality should be considered for inclusion in future HPF releases.

6 Programming Tools

A tool called PRISM is available for developing and optimizing CM Fortran codes [28]. The PRISM programming environment is integrated and graphical within which users can perform editing, debugging, data visualization, and performance analysis. It provides an easy-to-use, flexible, and comprehensive set of tools for performing all aspects of CM-5 programming for CM Fortran programs. No such tool is available for HPF programs. At this point, we discuss the functionality of PRISM and give a work around for HPF programs until a tool is available for HPF programs.

6.1 Editing and Compiling

One can edit and compile source code by invoking the appropriate utilities from PRISM. Currently, no such feature is available in any HPF compiler. The D editor developed by Ken Kennedy's group at Rice University may be used as a starting point to develop the one for HPF [37].

6.2 Debugging a Program

Currently, no HPF debugger is available. However, for `pghpf` compiler [29-30] the user can debug the generated Single Program Multiple Data (SPMD) Fortran 77 program on node using multiple X windows. Although very inconvenient, this work around for debugging is very useful to obtain a trace back of program that is crashing unexpectedly. This type of debugging can be done either on a processor or on multiple processors. The advantage of debugging HPF code on a single processor is that one can use standard debugger on the executable. We recommend that vendors provide the HPF debugger as soon as possible. This debugger need not be very sophisticated in the beginning.

6.3 Profiling a Program

Currently, no profiling tool is available for HPF codes. However, PGI has developed a graphical HPF profiling tool that does subroutine-level and line-level profiling of HPF codes. This tool is undergoing a Beta testing and may be available soon on some machines. At this time, profiling of HPF codes can be done manually by inserting a Fortran 90 intrinsic function `SYSTEM_CLOCK.` The resolution of this function is implementation dependent. The `pghpf` compiler provides a resolution of this intrinsic function as one microsecond. It is recommended that other HPF compiler vendors also provide the same resolution.

6.4 Visualizing Data

One can visualize either variables (including arrays, structures, pointers, *etc.*) or expressions of CM Fortran codes using PRISM. One can also provide a context, so that PRISM handles the values of data elements differently, depending on whether they meet the condition specified.

Lack of good programming tools are slowing the development of new codes in HPF. Parallel HPF debugger should be made available as soon as possible for HPF to become more acceptable to users.

6.5 HPF Automated Translator

On CM-5, a tool called CM Automated Translator (CMAX) [31] is available to convert Fortran 77 programs into CM Fortran. It is very desirable that a similar tool, HPF Automated Translator, be made available to the users for converting Fortran 77 codes into HPF codes. This tool need not be especially efficient to start but should provide some baseline HPF code. This tool should be separate from the HPF compiler.

7 HPF Compilers

HPF compilers can be classified by two classes:

7.1 Subset HPF Compilers

Subset HPF [22] is intended by HPF Forum to include a minimum set of features from Fortran 90 and HPF to encourage and facilitate an early release of production quality compilers. Vendors may include more features than listed in Subset HPF. Although not required by Subset HPF, PGI [29-30] has included `KIND`, Namelist I/O, *etc.* in their `pghpf` compiler. Currently, two Subset HPF compilers one from PG [29] and other from Applied Parallel Research (APR) [32] are available on NAS IBM SP2 and the HPCCP workstation cluster. The `pghpf` compiler has limited CM Fortran compatibility as follows:

a. It supports both the CM Fortran and HPF names. For example, CM Fortran / HPF: `DSHAPE/ SHAPE, DOTPRODUCT/DOT_PRODUCT, DLBOUND/LBOUND, DUBOUND/UBOUND.`

b. It supports both the CM Fortran and HPF versions of array constructors. It accepts the CM Fortran method of square brackets in the definition of array constructors, *i.e.*, a = `[1,2,3,4,5,6,7,8]` is a acceptable syntax. The corresponding syntax in HPF is `a = (/1,2,3,4,5,6,7,8/)`.

c. It supports CM Fortran `ARRAY` keyword in place of the Fortran 90 standard `DIMENSION` keyword as in:

CM Fortran: `INTEGER,ARRAY(m)::a,b;`
HPF: `INTEGER,DIMENSION(m)::a, b`

d. The three CM Fortran intrinsic functions CSHIFT, EOSHIFT and RESHAPE have names identical to corresponding functions in Fortran 90, but different calling sequences. Porting of these functions from CMF to HPF is shown in Figures 5 and 6. In HPF, boundary argument in EOSHIFT is not yet implemented.

```
y = CSHIFT(x, DIM=1, SHIFT=-1)
y = CSHIFT(x, 1, -1)

b=EOSHIFT(a,DIM=1,SHIFT=-1,BOUNDARY=[1:4]
b=EOSHIFT(a, 1, -1, [1:4])
```

Figure 5: Use of CSHIFT and EOSHIFT in CM Fortran.

```
y = CSHIFT(x, SHIFT=-1, DIM=1)
y = CSHIFT(x, -1, 1)

b=EOSHIFT(a,SHIFT=-1,BOUNDARY=[1:4],DIM=1)
b=EOSHIFT(a, -1, [1:4], 1)
```

Figure 6: Use of CSHIFT and EOSHIFT in HPF

e. The following intrinsics of CM Fortran: DIAGONAL, FIRSTLOC, LASTLOC, PROJECT, RANK and REPLICATE have no equivalent in Fortran 90 or HPF.

7.2 Full HPF Compiler

The DEC Fortran 90 V1.3 compiler the first to support both the ANSI/ISO Fortran 90 standards and the full *de-facto* HPF specification as claimed by DEC in a recent press release [33]. However, their claim is false as their compiler does not support `REALIGN, REDISTRIBUTE` or transcriptive mapping. The Digital Parallel Software Environment V1.0 provides key pieces of the software development and execution environment for parallel HPF applications. It includes program development tools like a debugger and profiler, as well as underpinnings such as optimized message passing and parallel system management tools. However, both are only available for DEC Alpha platforms running Digital OSF-1 and not on any other machines [33].

8 Results of Porting CM Fortran Codes to HPF

In this section, we will discuss the porting of CM Fortran codes from the CM-5 to HPF on the IBM SP2 and SGI Power Challenge.

8.1 Parallel Matrix Multiplication in HPF

The algorithm we have implemented is based on Cannon algorithm [34]. This algorithm was earlier implemented by the author on CM-5 using CM Fortran and on CRAY T3D using the CRAFT model [35]. It is implemented using only the Fortran 90 intrinsic function CSHIFT. CM Fortran implementation of this algorithm is shown in Figure 7, whereas HPF implementation is shown in Figure 8. In CM Fortran version, no layout directives are used to distribute the data as default data distribution is canonical distribution. However, in HPF version three directives (`TEMPLATE`, `DISTRIBUTE`, and `ALIGN`) are used to distribute and align the data. Performance and scalability of the HPF version of Cannon algorithm is compared with other parallel implementations and will be published separately [36].

8.2 Gaussian Elimination

The other code ported from CM Fortran to HPF solves systems of linear equations, *Ax= b,* using Gaussian elimination followed by backward substitution. Fortran 90 functions used in this program are `MAXLOC,SYSTEM_CLOCK,SELECTED_REAL_KIND,`and `RANDOM_NUMBER.`HPF functions or directives used are `PROCESSORS,NUMBER_OF_PROCESSORS(), TEMPLATE,ALIGN,`and `DISTRIBUTE.` We also used `FORALL` statement. We used various distribution schemes such `(block,block), (cyclic,cyclic),` and `(block,*)` to investigate the communication behavior and performance results will be published separately [36]. Speedup of this linear solver for A = 1000 x 1000 with blocked distribution as a function of processors for the IBM SP2 and SGI Power Challenge, and is given in Figure 9. Note that on both the IBM SP2 and SGI Power Challenge, performance first increases and then decreases. Results of these figures were obtained using the `pghpf` compiler.

8.3 Laplace Equation Solver

We solve Laplace's equation on a unit square by point Jacobi relaxation method, with the boundary condition that f = 1 on y = 1 and 2 elsewhere on the boundary, and the initial condition that f = 0 in the interior. The port from CM Fortran to HPF was straightforward. Various distribution schemes were tried and performance results (as a function of number of processors and different grid sizes) will be published separately [36].

8.4 Convolution

CM Fortran code performs a one-dimensional convolution over the first dimension of a two-dimensional dataset. This codes tests the performance of the Fortran intrinsic function EOSHIFT. The port of this code to HPF was straightforward. Once again, various distribution schemes were tried and performance results will be published elsewhere [36].

```
       PROGRAM nasmult
       PARAMETER (n = 1024)
       DOUBLE PRECISION, ARRAY(n,n) :: x, y, z, x1, y1
       INTEGER iskew(n)
       x = 2.0
       y = 3.0
       z = 0.0
c Multiply two square matrices x and y using Cannon’s algorithm
       FORALL (i =1:n) iskew(i) = i - 1
       x1 = CSHIFT( x, 2, iskew )
       y1 = CSHIFT( y, 1, iskew )
       z = 0.0d0
       DO ( n ) TIMES
        z = z + x1 * y1
        x1 = CSHIFT( x1, 2, 1 )
        y1 = CSHIFT( y1, 1, 1 )
       ENDDO
       PRINT *, z(1, 1), Z(n, n)
       z = 0.0d0
c Multiply two square matrices x and y using Fortran 90 library
       z = MATMUL(x, y)
       PRINT *, z(1, 1), z(n, n)
       END
```

Figure: 7 Parallel matrix multiplication in CM Fortran using Cannon algorithm.

```
       PROGRAM nasmult
       PARAMETER (n = 1024)
       DOUBLE PRECISION, DIMENSION(n,n) :: x, y, z, x1, y1
       INTEGER iskew(n)
!HPF$ TEMPLATE T(n,n)
!HPF$ DISTRIBUTE T(BLOCK,BLOCK)
!HPF$ ALIGN (:,:) WITH T :: Z
!HPF$ ALIGN (:,*) WITH T(:,*) :: x
!HPF$ ALIGN (*,:) WITH T(*,:) :: y
       x = 2.0
       y = 3.0
       z = 0.0
c Multiply two square matrices x and y using Cannon’s algorithm
       FORALL (i =1:n) iskew(i) = i - 1
       z = z + x1 * y1
       x1 = CSHIFT( x, 2, iskew )
       y1 = CSHIFT( y, 1, iskew )
       z = 0.0d0
       z = z + x1 * y1
       x1 = CSHIFT( x1, 2, 1 )
       y1 = CSHIFT( y1, 1, 1 )
       PRINT *, z(1, 1), Z(n, n)
       z = 0.0d0
c Multiply two square matrices x and y using Fortran 90 library
       z = MATMUL(x, y)
       PRINT *, z(1, 1), z(n, n)
       END
```

Figure: 8 Parallel matrix multiplication in HPF using Cannon algorithm.

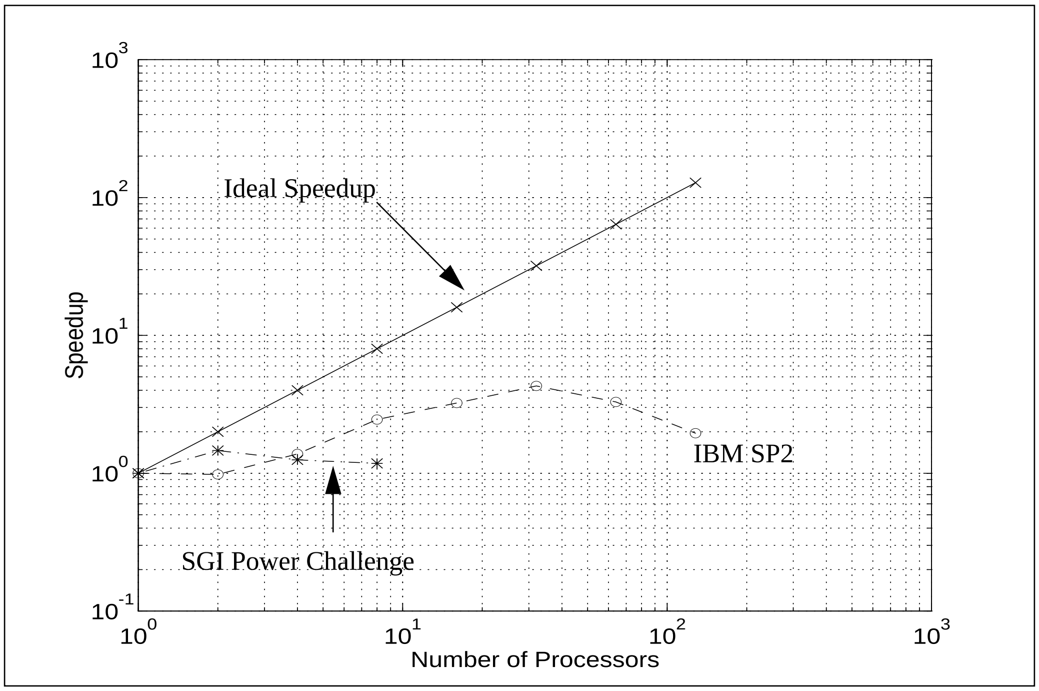

Figure 9: Speedup versus number of processors for solving Ax = b by Gaussian elimination.

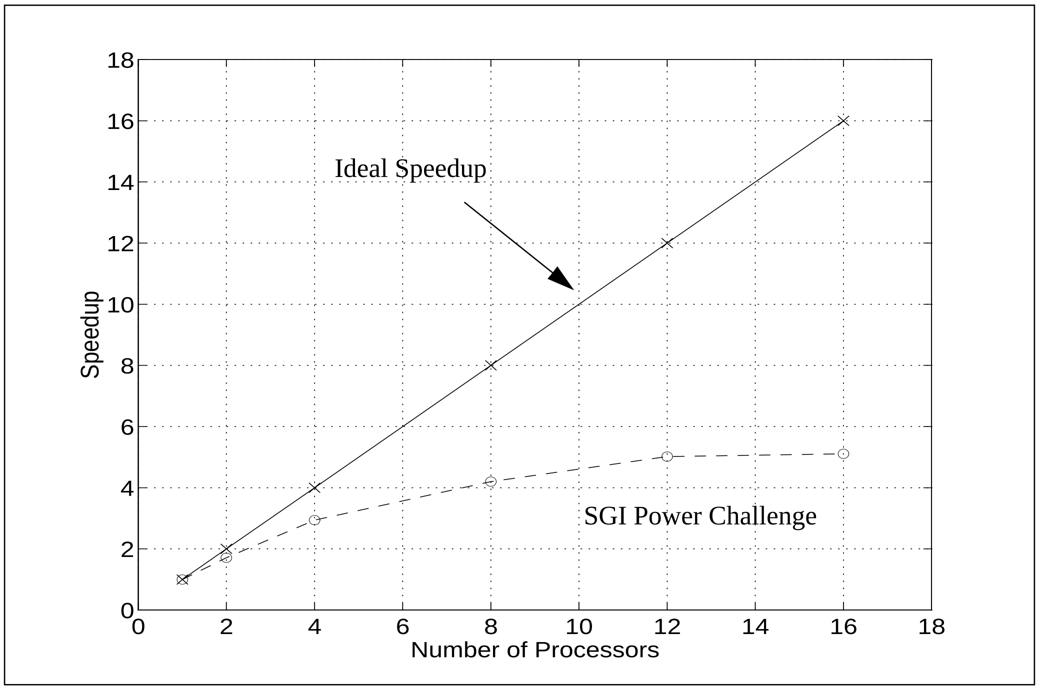

Figure 10: Speedup versus number of processors for DSMC code.

8.5 Direct Simulation Monte Carlo (DSMC)

Particle simulation using direct simulation Monte Carlo (DSMC) is a technique for analyzing low density flows and is routinely used for performing engineering analysis of aerospace vehicles. In the CM Fortran implementation of the code, there is fine-grain parallelism associated with the particles in the simulation and coarse-grain parallelism associated with the cells used in the simulation [38]. NAS, in collaboration with Portland Group has ported this CM Fortran code to HPF. Speedup of HPF code as a function of number of processors for the SGI Power Challenge is shown in Figure 10. Up to 12 processors, the code scales well. Beyond 12 processors, performance is almost flat. This is due to inadequate bus bandwidth of 1.2 GB/s for the SGI Power Challenge.

9. Deficiencies of Current HPF Compilers

Some of the problems encountered with specific HPF compilers are summarized as follows:

9.1 Portland Group HPF Compiler

a. Currently, parallelism is achieved only through array assignment, WHERE statement, WHERE construct, and FORALL statement.

b. Currently, the INDEPENDENT directive does not work.

c. DO loops are not parallelized; instead, they should be expressed as array assignments or FORALL statements should be used.

d. Performance of transformational intrinsics (SUM,SPREAD,TRANSPOSE, *etc.*) is poor.

e. Three CM Fortran intrinsics (CSHIFT, EOSHIFT, and RESHAPE) have identical names as in Fortran 90 but different calling sequences.

f. Most of the HPF codes do not compile with -O3 or -O2. This is a compiler bug. The absence of -O3 functionality degrades application performance.

g. The HPF intrinsic function PROCESSORS_SHAPE is not yet supported.

h. Intrinsic functions such as MATMUL needs to be optimized for each architecture.

9.2 Applied Parallel Research HPF Compiler

a. `DO` loops are not parallelized by the HPF directive INDEPENDENT. However, APR inserts its own directives for different iterations, to be spread over different processors.

b. In APR, intrinsic subroutines of Fortran 90 that are not yet available: `random_number`, `system_clock`, and `select_real_kind`.

d. Many Fortran intrinsics (`CSHIFT`, `EOSHIFT`, and `RESHAPE`) have identical names as in Fortran 90 but different calling sequences.

10 Conclusions

a. The porting of CM Fortran codes to HPF on the IBM SP2 and SGI Power Challenge machines proved to be straightforward. Much of the effort was devoted to translating CM Fortran layout and `ALIGN` directives to equivalent HPF distribute and align directives. Some time was also spent in replacing CM Fortran Utility routines with HPF equivalent functions.

b. It is recommended to vendors that they make canonical data distribution and canonical align as defaults in their HPF compilers.

c. Vendors should also provide `FORALL` construct for HPF compilers as soon as possible. Inclusion of `FORALL` construct will facilitate the porting of large numbers of CM Fortran codes to HPF with ease. Basic profiler and debugger tools should also be provided for HPF programs.

d. One of strong feature of CM Fortran is the ease with which one can perform parallel I/O. It is recommended to the HPF Forum to include parallel I/O consistent with CM Fortran. The capability of a communication compiler should also be considered for inclusion in the next HPF revision.

e. CM Fortran codes relying on CMSSL library could not be ported to HPF and this led to a lot of frustration. One of the biggest hurdles the NAS users are facing in porting CM Fortran codes to HPF on the IBM SP2 is lack of parallel mathematical and scientific libraries, which can be used and called from HPF codes. It has been argued in some quarters that the use of CMSSL library points to the poor quality of the CM Fortran compiler. On the CM-2 and CM-5, a CM Fortran Utility Library was available to work around for CMF compiler deficiencies. Eventually, most routines (including routines for parallel I/O) of this utility library became part of the matured CM Fortran compiler. Just as BLAS 2, BLAS 3, LIBSCI, and LAPACK are essential for successful parallel-vector programming, parallel scientific libraries will also be needed for the success of HPF on highly parallel computers. An immediate need for NAS HPF users is parallel Fast Fourier Transform, block tridiagonal, and block pentadigonal solvers. We recommend to vendors that CMSSL is a logic library, to emulate based on the fact that CMSSL is a industry leader and the production CM Fortran codes far exceeds that of any other vendor. One of most powerful feature, of CMSSL is “multiple instances.” This feature permit users to perform multiple independent operations on different datasets concurrently.

f. It is our belief that vendors should pay special attention for inclusion of an efficient data parallel programming model as defined by the HPF Forum as a paradigm characterized by single-thread execution, global name space, and loose synchronously. Although NAS recognizes that HPF may not be a ultimate solution for perfect parallel programming, it seems to offer the best hope for maintaining a single portable version of codes on all computing platforms.

g. The success of HPF hinges on the development of compilers that can provide reasonable performance satisfactory to users. The first and foremost task for HPF compiler developers should be to develop a HPF compiler that will generate code equal to or better than CM Fortran with consistently high performance across all computing platforms. As the vendors move closer to that goal, they should consider taking a more active role in the development of programming tools, as well.

h. Computations such as FFT, banded tridiagonal solvers, finite element, n-body problems possess regular but specific data access patterns that HPF compilers need to recognize and support.

Acknowledgments

I would like to thank David H. Bailey of NAS-NASA Ames Research Center and Robert Schreiber of RIACS at NAS-NASA Ames Research Center for carefully reading this manuscript and for offering several valuable suggestions.

References

[1] High Performance Fortran Forum, *High Performance Fortran Language Specification,* Version 1.0, January 1993.

[2] S. Saini, “High Performance Fortran,” *NAS User Televideo Seminar,* August 17, 1993. Videotape and notes available from `doc-center@nas.nasa.gov`.

[3] S. Saini, “High Performance Fortran - Part I,” *NAS Newsletter,* NASA Ames Research Center, January 1994.

[4] S. Saini, “High Performance Fortran - Part II,” *NAS Newsletter*, NASA Ames Research Center, March 1994.

[5] *Intel iPSC/860 User’s Guide,* April, 1993, Intel Corporation.

[6] *Intel Paragon OSF/1 User Guide,* Intel Corporation, 1994.

[7] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam, *PVM 3 User’s Guide and Reference Manual*, Oak Ridge National Laboratory ORNL/TM-12187, May 1993.

[8] *Message Passing Interface Forum MPI: A Message-Passing Interface Standard,* Computer Science Dept. Technical Report CS-94-230, University of Tennessee, April 1994.

[9] *CM Fortran User’s Guide*, Version 2.1, January 1994. Thinking Machines Corporation.

[10] *ANSI X3J3/S8.115, Fortran 90,* June 1990.

[11] G. Fox, S. Hiranandani, K. Kennedy, C. Koelbel, U. Kremer, C. W. Tseng, and M. Mu, F*ortran D Language Specification, Technical Report TR90-141*, Department of Computer Science, Rice University, December 1990.

[12] B. Chapman, P. Mehrotra, and H. Zima, “Programming in Vienna Fortran,” *Scientific Programming*, 1(1), pp. 31-50, Fall 1992.

[13] D M. Pase, T. MacDonald, A. Meltzer, “The CRAFT Fortran Programming Model,” *Scientific Programming*, Volume 3, pp. 227-253, 1994.

[14] S. Saini, H. D. Simon, and C. Grassl, “High Performance Programming Using Explicit Shared Memory Model on the CRAY T3D”, Technical Report RNR-94-012, May 1994, Applied Research Branch, NASA Ames Research Center. Also published in Cray User Group Spring 1994 Meeting, San Diego, California.

[15] S. Saini and H. D. Simon, “High Performance Computing for Scientific Applications: An Introduction,” T*utorial M7, Supercomputing ‘94*, November 14, 1994, Washington D.C.

[16] S. Saini, “The HPCCP Cluster: Hardware and Software Overview of SGI Power Challenge,” *NAS Televideo Seminar,* Oct. 1994. Report and videotape available from `doc-center@nas.nasa.gov.`

[17] *MIPSpro POWER Fortran 77 Programmer’s Guide,* 1994, Silicon Graphics Corporation.

[18] *Exemplar Programming Guide,* April 1993, Convex DSW-067, Convex Computer Corporation.

[19] *UXP/M VPP FORTRAN77 EX/VPP USER’S GUIDE (V12),* # 94SP5100E-1, First Edition, October 1994, Fujitsu Limited, Japan.

[20] S. Saini and D. H. Bailey, “NAS Parallel Benchmark Results,” *Submitted to Supercomputing ‘95*. Also available as NAS Technical Report “NAS Parallel Benchmarks Results - March 1995.” This report can be obtained from `doc-center@nas.nasa.gov`.

[21] S. Saini, “Early NAS Experiences of HPF Compilers on IBM SP2 and the HPCCP SGI Cluster,” *Proceedings of High Performance Fortran Forum Annual Meeting*, Houston, January 30-31, 1995.

[22] C. H. Koelbel, D. B. Loveman, R. S. Schreiber, G. L. Steele, and M. E. Zosel, *The High Performance Fortran Handbook*, Scientific and Engineering Computation Series, The MIT Press.

[23] *CM-5 CM Fortran Performance Guide*, Version 2.1, January 1994, Thinking Machines Corporation.

[24] *CM Fortran Utility Library Reference Manual*, Version 2.0, January 1993, Thinking Machines Corporation.

[25] *CMSSL for CM Fortran: CM-5*, Volume I and II, Version 3.1, June 1993 Edition.

[26] *CM-5 CM Fortran User Guide*, Version 2.1, Thinking Machines Corporation.

[27] S. Saini, “The IBM SP2 Supercomputer: Hardware and Software Overview,” *NAS Telvideo User Seminar,* July 27, 1995. Report and videotape are available from `doc-center@nas.nasa.gov`.

[28] *PRISM User’s Guide*, Version 2.0, April 1994, Thinking Machines Corporation.

[29] *PGHPF User’s Guide,* November 1994, The Portland Group Inc.

[30] *PGHPF Reference Manual,* November 1994, The Portland Group Inc.

[31] *Using the CMAX Converter*, Version 2.0, May 1994, Thinking Machines Corporation.

[32] *FORGE High Performance Fortran - HPF Parallelizing Pre-Compiler for Distributed Memory Multi-Processor Systems - xHPF,* Version 2.0, User Guide, January 1995 Release, Applied Parallel Research Inc.

[33] “Digital Debuts High Performance Fortran/Parallel Computing Tool,” *HPCwire # 7048,* 1995.

[34] L. E. Cannon, “A Cellular Computer To Implement the Kalman Filter Algorithm,” Ph.D Thesis, Montana State University, 1969.

[35] S. Saini, “High Performance Programming Using Explicit Shared Memory Model on the CRAY T3D,” *NAS Televideo User Seminar,* January 19, 1994. Report and videotape are available from `doc-center@nas.nasa.gov`

[36] S. Saini, to be published 1995.

[37] S. Hiranandani, K. Kennedy, C. W. Tseng, and S. Warren, “The D Editor: A New Interactive Parallel Programming Tool,” pp. 733-742, *Proceedings Supercomputing ‘94*, Washington D.C.

[38] L. Dagum. “Three-Dimensional Direct Particle Simulation on the Connection Machine,” Journal of Thermophysics and Heat Transfer, Vol. 6, No. 4, Oct.-Dec. 192

[39] D. H. Bailey, S. Saini, and R. S. Schreiber, *et al.* - A group has been formed to identify and develop scientific libraries needed by NAS users in HPF codes.